IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
Before the Board of Patent Appeals and Interferences

In re Patent Application of
NEVILL
Serial No. 09/726,391
Filed: December 1, 2000
Title: INSTRUCTION INTERPRETATION WITHIN A DATA PROCESSING SYSTEM

Atty Dkt. 550-189
C# M#
TC/A.U.: 2195
Examiner: Lewis A. Bullock, Jr.
Date: November 7, 2005

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

☐ **Correspondence Address Indication Form Attached.**

☐ **NOTICE OF APPEAL**
Applicant hereby **appeals** to the Board of Patent Appeals and Interferences from the last decision of the Examiner twice/finally rejecting applicant's claim(s). $500.00 (1401)/$250.00 (2401) $

☒ A **REINSTATED APPEAL BRIEF** is attached in the pending appeal of the above-identified application
$500.00 (1402)/$250.00 (2402) $

☐ Credit for fees paid in prior appeal without decision on merits -$ ( )

☐ A reply brief is attached. **(no fee)**

☒ Petition is hereby made to extend the current due date so as to cover the filing date of this paper and attachment(s)
One Month Extension $120.00 (1251)/$60.00 (2251)
Two Month Extensions $450.00 (1252)/$225.00 (2252)
Three Month Extensions $1020.00 (1253/$510.00 (2253)
Four Month Extensions $1590.00 (1254/$795.00 (2254) $ 450.00

☐ "Small entity" statement attached.

Less month extension previously paid on -$( )

TOTAL FEE ENCLOSED $ 450.00

Any future submission requiring an extension of time is hereby stated to include a petition for such time extension. The Commissioner is hereby authorized to charge any deficiency, or credit any overpayment, in the fee(s) filed, or asserted to be filed, or which should have been filed herewith (or with any paper hereafter filed in this application by this firm) to our **Account No. 14-1140.** A duplicate copy of this sheet is attached.

11/08/2005 JADD01 00000004 09726391
NIXON & VANDERHYE PC 252 450.00 OP

901 North Glebe Road, 11th Floor
Arlington, Virginia 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100
JRL:sd

By Atty: John R. Lastova, Reg. No. 33,149

Signature: _____

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

NEVILL — Atty. Ref.: 550-189

Serial No. 09/726,391 — Group: 2195

Filed: December 1, 2000 — Examiner: Bullock Jr, Lewis A.

For: INSTRUCTION INTERPRETATION WITHIN A DATA PROCESSING SYSTEM

Before the Board of Patent Appeals and Interferences

BRIEF FOR APPELLANT

Reinstated Appeal From Rejection
By Group Art Unit 2195

John R. Lastova
NIXON & VANDERHYE P.C.
11th Floor, 901 North Glebe Road
Arlington, Virginia 22203-1808
(703) 816-4000
Attorney for Appellant
Nevill
ARM Limited

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Patent Application of

NEVILL — Atty. Ref.: 550-189

Serial No. 09/726,391 — Group: 2195

Filed: December 1, 2000 — Examiner: Bullock Jr, Lewis A.

For: INSTRUCTION INTERPRETATION WITHIN A DATA PROCESSING SYSTEM

* *

November 7, 2005

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

Sir:

I. REAL PARTY IN INTEREST

The real party in interest is the assignee, ARM Limited, a United Kingdom corporation.

II. RELATED APPEALS AND INTERFERENCES

This is a reinstatement of the appeal with the brief filed on April 18, 2005. There are no other appeals related to this subject application. There are no interferences related to this subject application.

III. STATUS OF CLAIMS

Claims 1-14 are pending. Claims 1-7 and 9-14 stand rejected under 35 U.S.C. §102 for anticipation based on USP 6,065,108 to Tremblay et al. Claim 8 stands rejected under 35 U.S.C. §103 as being unpatentable over Tremblay et al.

IV. STATUS OF AMENDMENTS

No amendment has been filed after final.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

The claims are directed to data processing systems that have an instruction interpreter that replaces a slow form instruction with a fast form instruction and that operates using a separate instruction store and data store. An example of a data processing system that operates using a separate instruction store and data store is often referred to as a Harvard architecture. An example of a separate data store and an instruction store in the form of a separate data cache and instruction cache is illustrated in Figure 1 reproduced here for convenience:

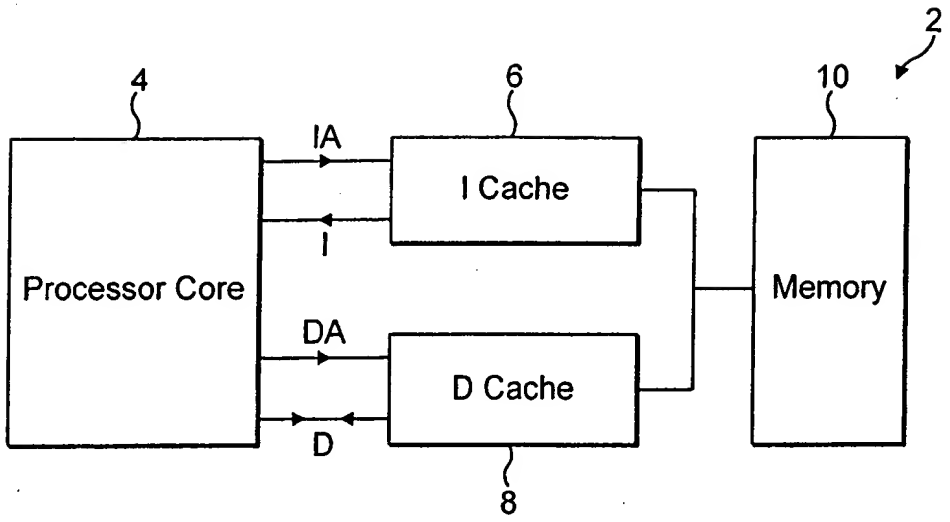


FIG. 1

In the operation of the data processing system 2, instructions to be executed are read from the main memory 10 into the instruction cache 6 and then from the instruction cache 6 into the processor core 4 where they are executed. Data words required for the data processing operation specified by the instructions or generated by those instructions are read from or written to the data cache 8.

One problem with this arrangement is how to deal with instruction code that is dynamically altered at runtime. It is known to provide an instruction interpreter that will modify a slow form of instruction to a fast form of instruction at runtime. In a Harvard architecture, the instructions are typically provided within a read only instruction store, and the writing of a modified fast form of instruction out to the data store requires a flush and reload of at least some portions of the data and instruction stores to avoid a risk of inconsistency between different forms of the same instruction being held in the instruction store and the data store. But this flush and reload reduces the system performance.

This problem is solved by having the instruction interpreter check, upon encountering a slow form instruction, whether a corresponding fast form instruction exists within the data store. If present, the interpreter replaces the slow form instruction with that fast form instruction. The slow form instruction and the fast form instruction have a common functionality when executed by the interpreter. The inventor discovered that the additional processing overhead associated with this check within the data store for a fast form of instruction is more than compensated for by the ability reliably to replace slow form instructions with fast form instructions with systems having a separate data store and instruction store. Figure 2 illustrates an example procedure for such checking (block 12) and replacement (block 14) if the corresponding fast form instruction already exists within the data store. Otherwise, a resolution procedure is performed to generate a corresponding fast form instruction (see blocks 18 and 20).

As set forth in the independent claims 1 and 14, even though the instruction interpreter can modify a slow form instruction within the instruction store to one or more possible fast form instructions and can write the fast form instruction to the data store, the instruction interpreter first checks a slow form instruction read from the instruction store for a corresponding fast form instruction already stored within the data store. If the fast form instruction is present within the data store, then the interpreter executes the fast form instruction.

An advantageous example application is one in which an unresolved memory access is dynamically replaced by a resolved memory access. The unresolved memory access typically involves a symbolic reference to the data or instructions being sought. The resolved memory access typically includes a numeric reference to this information.

A numeric reference can be directly used to return the required information, which greatly increases processing speed. Another advantageous application is a situation in which a slow form instruction invokes additional processing procedures before completion, such as calls to further processing resources, that may even be on remote systems. And as the detailed example embodiment described at page 6, line 10-page 10, line 10 and illustrated in Figures 3 and 4 demonstrates, the ability to properly replace a slow form instruction with a fast form instruction is particularly useful when interpreting Java Virtual Machine instructions.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The rejections to be reviewed on appeal are the anticipation and obviousness rejections based on Tremblay et al.

VII. ARGUMENT

A. Tremblay Fails to Disclose Every Feature in Claims 1 and 14

To establish that a claim is anticipated, the Examiner must point out where each and every limitation in the claim is found in a single prior art reference. *Scripps Clinic & Research Found. v. Genentec, Inc.*, 927 F.2d 1565 (Fed. Cir. 1991). Every limitation contained in the claims must be present in the reference, and if even one limitation is missing from the reference, then it does not anticipate the claim. *Kloster Speedsteel AB v. Crucible, Inc.*, 793 F.2d 1565 (Fed. Cir. 1986). Tremblay fails to satisfy this rigorous standard.

B. The Tremblay Reference

Tremblay relates to the accelerated execution of instructions in a computer system. Many computer systems, including those implementing the JAVA virtual machine specification, implement instructions which require that additional data be located and retrieved prior to executing the instruction. Instructions which require that data be located and retrieved prior to execution are referred to by Tremblay as "non-quick instructions." The problem Tremblay identifies is that locating and retrieving the data required to execute a non-quick instruction can take hundreds of cycles. Tremblay explicitly wants to accelerate execution of non-quick instructions but without having to modify non-quick instructions. See column 3, lines 17-28.

So rather than modifying the non-quick instructions to make them quicker, Tremblay uses instruction identifiers associated with non-quick instruction that have a associated data set that must be accessed prior to executing the non-quick instruction. An associative memory 14 stores instruction identifiers (corresponding program counter values) and the associated data set. When a current instruction identifier value in a stream of instruction identifier values matches an instruction identifier value stored in the memory, an associated data set is accessed from the memory. More specifically, if a first instruction identifier value and the first data set are stored in the memory, and the current instruction identifier value is equal to the first instruction identifier value, then the first data set is read out of the memory. Execution of the non-quick instruction is accelerated because the first data set is readily accessible within the memory.

C. Temblay's Accelerator Does Not Modify a Slow Form Instruction into a Fast Form Instruction and Write It to a Data Store

Claims 1 and 14 recite features not described by Tremblay. For example, Tremblay fails to disclose integer (vi) of claim 1 where *"said instruction interpreter is operable to modify a slow form instruction within said instruction store to a fast form instruction of one or more possible fast form instructions and to write said fast form instruction to said data store..."* A similar recitation is found in claim 14.

There is a fundamental difference between what is recited in claims 1 and 14 and the system of Tremblay: In Tremblay's acceleration process, the non-quick instruction emains in its original form in memory and is not overwritten. See col. 4, lines 10-15: "Because the non-quick instruction is not overwritten, the non-quick instruction remains available in its original form. Moreover, because the non-quick instruction is not overwritten, the non-quick instruction can be stored in read only memory." See similar language at col. 28, lines 35-40. In contrast, the instruction acceleration in claims 1 and 14 modifies a slow-form instruction from an instruction store by an instruction interpreter to a fast-form instruction, and that fast form instruction is written to a data store. The stored fast-form instruction is subsequently executed in place of the corresponding slow-form instruction.

To illustrate this difference, consider the following example based on the detailed embodiment described in the instant specification. The claims are not limited to this example or this embodiment. The example illustrates the data stored by a data processing after initial execution of a non-quick instruction which in this case is a "getfield" instruction:

Data Stored By Example Embodiment Of Present Application After Initial Execution Of Getfield Non-Quick Instruction

Instruction Identifier	Instruction Store	Data Store
PC = 0	getfield <data set>	getfield_quick <resolved data set>
PC = 3		

Main or external memory	
PC = 0	Getfield <data set>

Data Stored By System Of Tremblay After Initial Execution Of "Getfield" Non-Quick Instruction

Instruction identifier	Memory
PC = 0	getfield <data set>
PC = 3	

Associative Memory 14 in Fig 6	
Instruction identifier	Data set
PC = 0	<resolved data set>

This example shows that Tremblay does not write a fast-form instruction to a data store, as specified by claims 1 and 14, but simply stores a data set and instruction identifier (the program counter (PC) value) associated with the slow-form instruction.

In the response to argument section of the official action, the Examiner argues that col. 27, line 5 to col. 28, line 6 and col. 28, line 48 to col. 29, line 14 of Tremblay disclose an interpreter modifying a slow form instruction to a fast form instruction and writes the fast form instruction to the data store. This is not the case.

Col. 27, line 5 to col. 28, line 6 discloses that software code portions 31, 32, and 33 are used to retrieve data sets for storage in the data set memory 20 of the associative memory 14 to enable a non-quick instruction associated with an instruction identifier to

be executed as a quick instruction. Although each data set is *associated with* a non-quick instruction, the data set is not a fast form instruction.

Col. 28, line 48 to col. 29, line 14 discloses that an instruction identifier is used to determine that an instruction is a non-quick instruction having a "quick variant", and trap logic initiates execution of a software portion to retrieve the data set required to execute the instruction. The retrieved data set is loaded into an operand stack 423. Quick variants are not defined, but the use of the term at the bottom of col. 14 and the top of column 15 suggests that is simply a shorthand way of referring to Tremblay's way of indicating that a non-quick instruction can be accelerated using its corresponding instruction identifier (program counter value) and associated data set stored in the associative memory 14. This passage of Tremblay neither discloses nor suggests that a fast-form instruction that has been formed by modifying a slow-form instruction is written to a data store as specified by claims 1 and 14.

D. Tremblay Does Not Check for a Corresponding Fast Form Instruction in the Data Store

Tremblay does not disclose the subject-matter of claim 1 integer (vii) whereby the instruction interpreter is operable to check for a corresponding fast form instruction within the data store. For Tremblay's non-quick instructions which are not modified and remain in tact, there are no corresponding fast form *instructions* stored in the associative memory 14. Again, an instruction identifier is not the instruction; nor is the associated data set. Neither the identifier nor the data set can be executed like an instruction. Claim 14 has a similar recitation.

Thus, Tremblay lacks two of the integers recited in the independent claims.

E. Tremblay Fails to Disclose the Claimed Storage and Processing Architecture

Tremblay also fails to disclose the main memory, data store, and instruction store arrangement recited in claims 1 and 14. For example, claim 1 specifies a memory structure (effectively a Harvard architecture) where the data store and the instruction store are each operable to store words from the main memory and are accessed via respective ports to the processor core. Tremblay does not disclose that claimed memory structure.

The Examiner contends that Tremblay's associative memory 14 is a counterpart of the claimed main memory. The Examiner also contends that the data set memory section 20 of the associative memory 14 corresponds to the claimed data store claim 1 and that the instruction identifier memory section 18 of the associative memory 14 corresponds to the claimed instruction store 1. But the claims recite that each of the instruction store and the data store is operable to store words "*from said main memory.*" The associative memory 14 of Tremblay cannot be both the claimed main memory as well as the claimed instruction store and the data store.

F. Tremblay Fails to Render Obvious The Claims

Although the system of Tremblay and the claimed apparatus and method both deal with accelerating non-quick instructions, their approaches are fundamentally different. Tremblay stores an instruction identifier value and a corresponding data set in

an associative memory and performs a check on the identifier values in order to execute a fast form instruction. In contrast, the claimed approach stores a modified instruction in the data store and performs a check for the presence of the modified instruction.

The claimed approach exploits the claimed memory structure (Harvard architecture) where a data store and an instruction store are each operable to store words from the main memory and are accessed via respective ports to the processor core. A slow form instruction from the instruction store is modified to a fast form instruction and the modified instruction is written to the data store. When the instruction interpreter reads a slow form instruction from the instruction store it checks for a corresponding fast form instruction in the data store.

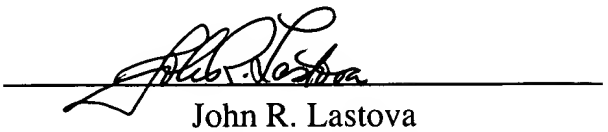
An advantage with regard to Tremblay that there is no need to provide an additional associative memory 14 in order to accelerate non-quick instructions. Rather, the dual nature of the Harvard architecture is exploited to provide a mapping between non-quick instructions stored in the instruction store and their corresponding modified forms (quick instructions) stored in the data store.

VIII. CONCLUSION

Tremblay's failure to teach multiple features of the claims and Tremblay's fundamentally different approach to accelerating slow form instructions requires reversal of the outstanding rejections.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: 

John R. Lastova
Reg. No. 33,149

JRL/kmm
Enclosures
Appendix A - Claims on Appeal

IX. CLAIMS APPENDIX

1. Apparatus for processing data, said apparatus comprising:

(i) a processor core;

(ii) a main memory operable to store instruction words and data words;

(iii) a data store operable to store words from said main memory accessed by a data store port of said processor core;

(iv) an instruction store operable to store words from said main memory accessed by an instruction store port of said processor core; and

(v) an instruction interpreter operable to read instruction words from said instruction store; wherein

(vi) said instruction interpreter is operable to modify a slow form instruction within said instruction store to a fast form instruction of one or more possible fast form instructions and to write said fast form instruction to said data store, said slow form instruction and said fast form instruction having a common functionality when executed by said interpreter; and

(vii) said instruction interpreter is operable upon reading a slow form instruction from said instruction store to check for a corresponding fast form instruction within said data store and, if said fast form instruction is present within said data store, then to execute said fast form instruction instead of said slow form instruction.

2. Apparatus as claimed in claim 1, wherein said instruction interpreter is a hardware based instruction translator.

3. Apparatus as claimed in claim 1, wherein said instruction interpreter is a software based interpreter.

4. Apparatus as claimed in claim 1, wherein said instruction interpreter is a combination of a hardware based instruction translator and a software based interpreter.

5. Apparatus as claimed in claim 1, wherein said data store is a data cache and said data store port is a data cache port.

6. Apparatus as claimed in claim 1, wherein said instruction store is an instruction cache and said instruction store port is an instruction cache port.

7. Apparatus as claimed in claim 1, wherein said slow form instruction results in an unresolved storage access request to one or more stored words and said fast form instruction results in a resolved storage access request to said one or more stored words.

8. Apparatus as claimed in claim 1, wherein said slow form instruction includes a symbolic reference to a required element and said fast form instruction includes a numeric reference to said required element.

9. Apparatus as claimed in claim 1, wherein said slow form instruction invokes an additional data processing procedure before completion.

10. Apparatus as claimed in claim 1, wherein said slow form instruction and said fast form instruction are Java Virtual Machine instructions.

11. Apparatus as claimed in claim 10, wherein said slow form instruction is one of:

anewarray;
checkcast;
getfield;
getstatic;
instanceof;
invokeinterface;
invokespecial;
invokestatic;
invokevirtual;
ldc;
ldc_w;
ldc2_w;
multianewarray;

new;

putfield; and

putstatic.

12. Apparatus as claimed in claim 10, wherein said fast form instruction is one of:

anewarray_quick;

checkcast_quick;

getfield_quick;

getfield_quick_w;

getfield2_quick;

getstatic_quick;

getstatic2_quick;

instanceof_quick;

invokeinterface_quick;

invokenonvirtual_quick;

invokesuper_quick;

invokestatic_quick;

invokevirtual_quick;

invokevirtual_quick_w;

invokevirtualobject_quick;

ldc_quick;

ldc_w_quick;

ldc2_w_quick;

multianewarray_quick;

new_quick;

putfield_quick;

putfield_quick_w;

putfield2_quick;

putstatic_quick; and

putstatic2_quick.

13. Apparatus as claimed in claim 10, wherein said instruction interpreter translates Java Virtual Machine instructions to native instructions of said processor core.

14. A method of processing data using an apparatus having a processor core, a main memory operable to store instruction words and data words, a data store operable to store words from said main memory accessed by a data store port of said processor core, an instruction store operable to store words from said main memory accessed by an instruction store port of said processor core, and an instruction interpreter operable to read instruction words from said instruction store; said method comprising the steps of:

(i) modifying a slow form instruction within said instruction store to a fast form instruction of one or more possible fast form instructions and to write said fast form instruction to said data store, said slow form instruction and said fast form instruction having a common functionality when executed by said interpreter; and

(ii) upon reading a slow form instruction from said instruction store, checking for a corresponding fast form instruction within said data store and, if said fast form instruction is present within said data store, then executing said fast form instruction instead of said slow form instruction.

X. EVIDENCE APPENDIX

There is no evidence appendix.

XI. RELATED PROCEEDINGS APPENDIX

There is no related proceedings appendix.